

Docket No. AUS920010942US1

**METHOD AND APPARATUS FOR IMPLEMENTING PERMISSION BASED
ACCESS CONTROL THROUGH PERMISSION TYPE INHERITANCE**

RELATED APPLICATION

5

This application is related to co-pending and commonly assigned U.S. Patent Application Serial No.

____ (Attorney Docket No. AUS920010941US1), entitled
"Method and Apparatus for Type Independent Permission
10 Based Access Control," filed on even date herewith and
hereby incorporated by reference.

BACKGROUND OF THE INVENTION

15 **1. Technical Field:**

The present invention is directed to an improved data processing system and, in particular, an improved mechanism for permission based access control. More specifically, the present invention provides a mechanism
20 through which a security provider may specify their own permission type and a permission type that can handle any application permission types.

2. Description of Related Art:

25 In Java Development Kit (JDK) version 1.1, local applications and correctly digitally signed applets were generally trusted to have full access to vital system resources, such as the file system. Unsigned applets were not trusted and could access only limited resources.
30 A security manager was responsible for determining which resource accesses were allowed.

In the Java 2 Software Developer's Kit (SDK), the

Docket No. AUS920010942US1

security architecture is policy-based and allows for fine-grained access control. In Java 2 SDK, when code is loaded by the classloader, it is assigned to a protection domain that contains a collection of "permissions" based on the security policy currently in effect. Each permission specifies a permitted access to a particular resource, such as "read" and "write" access to a specified file or directory, or "connect" access to a given host and port. The policy, specifying which permissions are available for code from various signers and/or locations, can be initialized from an external configurable policy file. Unless a permission is explicitly granted to code, it cannot access the resource that is protected by that permission. These concepts of permission and policy enable the Java 2 SDK to offer fine-grain, highly configurable, flexible, and extensible access control. Such access control can be specified not only for applets, but also for all Java code including but not limited to applications, Java beans, and servlets.

Because the Java 2 SDK provides such a flexible and extensible access control, Application vendors end up specifying different security policies in terms of their own permission types, thereby introducing their own type of permissions into the security policy. In order to provide access control functionality to these "customized" permissions, security provider implementors that create these custom permissions must make sure they understand these customized permission types and need to make sure they support them. For example, if the application code checks for "com.foo.security.AppPerm1", then the security provider needs to check

Docket No. AUS920010942US1

"com.foo.security.AppPerm1."

With evolving e-business and discrete sets of permissions getting introduced by many application providers, the number of permissions that need to be
5 handled grows exponentially. This also creates a tight coupling between the application provider and the security provider. This results in the security policy being placed in the application code, making code portability an issue. This tight coupling needs to be
10 avoided to provide seamless support to varied applications.

SUMMARY OF THE INVENTION

The present invention provides a method and apparatus for type independent permission based access control. The present invention solves the problems of the prior art by allowing security providers to specify their own permission type and a permission that can handle any application permissions below it in a hierarchy of permissions.

10 The present invention utilizes object inheritance that is built into the Java programming language to provide a mechanism by which a large group of permissions may be assigned to a codesource without having to explicitly assign each individual permission to the
15 codesource. With the present invention, a base permission class, also referred to as a superclass permission class, is defined from which other (sub) permissions inherit thus creating a hierarchy of permissions. Having defined the permissions in such a
20 hierarchy, a developer may assign a base or superclass permission to an installed codesource and thereby assign all of the inherited permissions of the base permission to the installed codesource. Thus, new applications may be installed and the developer or user need only assign
25 the base permission in order for the new application to work with other previously installed applications.

In this way, security providers need not know all the permission types defined in an application. In addition, security providers can seamlessly integrate
30 with many applications without changing their access control and policy store semantics. Moreover, application providers' security enforcement is not dependent on the security provider defined permissions.

Docket No. AUS920010942US1

The present invention does not require any changes to the Java Ssecurity Mmanager and does not require changes to application code.

5 These and other features and advantages of the present invention will be described in, or will become apparent to those of ordinary skill in the art in view of, the following detailed description of the preferred embodiments.

10

11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
221

BRIEF DESCRIPTION OF THE DRAWINGS

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself, however, as well as a preferred mode of use, further objectives and advantages thereof, will best be understood by reference to the following detailed description of an illustrative embodiment when read in conjunction with the accompanying drawings, wherein:

10 **Figure 1** is an exemplary diagram illustrating a distributed data processing system according to the present invention;

Figure 2 is an exemplary diagram of a server computing system according to the present invention;

15 **Figure 3** is an exemplary diagram of a client computing system according to the present invention;

Figure 4 is an exemplary diagram illustrating a Java Virtual Machine;

Figure 5 is a diagram illustrating a permission hierarchy according to the present invention;

20 **Figure 6** is a diagram illustrating an operation of the present invention when an untrusted resource access request is processed;

Figures 7A and 7B illustrate an exemplary superclass permission according to the present invention;

Figure 7C illustrates an exemplary subclass permission according to the present invention;

Figure 8 is an exemplary illustration of code for performing the operations of the present invention using the base or superclass permission class, IBMPermission and inherited or subclass permission WSPermission;

30 **Figure 9** is a flowchart outlining an exemplary

operation of the present invention for adding permissions to a permission collection;

Figure 11 is an example of code of a policy file according to the alternative embodiment.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

The present invention provides a mechanism by which permissions in an object oriented environment may be defined and used such that a single base permission, hereafter referred to as a superclass permission, may be assigned to a codesource and all inherited permissions, hereafter referred to as subclass permissions, falling under the base permission are thereby assigned to the codesource. In this way, a developer need not explicitly assign all of the possible permissions to the class but may assign only the base permission and obtain the benefit of any inherited permissions falling under the base permission in the permissions hierarchy.

The preferred embodiments of the present invention will be described with regard to the Java 2 SDK, however the present invention is not limited to use with Java 2 SDK. Rather, the present invention may be used with any object oriented environment in which inheritance and security policies are utilized to protect access to computer system resources. Thus, the references to Java 2 SDK and elements of this programming environment are only intended to be exemplary and are not intended to imply any limitation on the present invention.

Since the present invention operates in a Java security environment, a brief description of the Java computing environment will be provided. As is well known, Java is typically used in a client/server or other distributed data processing environment, although Java may also be used on a single computing device as well. As such, the following description of the Java computing environment will assume a client/server environment although the present invention may also be used by a

Docket No. AUS920010942US1

single computing device with or without a network connection.

With reference now to the figures, **Figure 1** depicts a pictorial representation of a network of data processing systems in which the present invention may be implemented. Network data processing system **100** is a network of computers in which the present invention may be implemented. Network data processing system **100** contains a network **102**, which is the medium used to provide communications links between various devices and computers connected together within network data processing system **100**. Network **102** may include connections, such as wire, wireless communication links, or fiber optic cables.

In the depicted example, server **104** is connected to network **102** along with storage unit **106**. In addition, clients **108**, **110**, and **112** are connected to network **102**. These clients **108**, **110**, and **112** may be, for example, personal computers or network computers. In the depicted example, server **104** provides data, such as boot files, operating system images, and applications to clients **108-112**. Clients **108**, **110**, and **112** are clients to server **104**. Network data processing system **100** may include additional servers, clients, and other devices not shown. In the depicted example, network data processing system **100** is the Internet with network **102** representing a worldwide collection of networks and gateways that use the TCP/IP suite of protocols to communicate with one another. At the heart of the Internet is a backbone of high-speed data communication lines between major nodes or host computers, consisting of thousands of commercial, government, educational and other computer systems that route data and messages. Of course, network data

Docket No. AUS920010942US1

processing system **100** also may be implemented as a number of different types of networks, such as for example, an intranet, a local area network (LAN), or a wide area network (WAN). **Figure 1** is intended as an example, and not
5 as an architectural limitation for the present invention.

Referring to **Figure 2**, a block diagram of a data processing system that may be implemented as a server, such as server **104** in **Figure 1**, is depicted in accordance with a preferred embodiment of the present invention.

10 Data processing system **200** may be a symmetric multiprocessor (SMP) system including a plurality of processors **202** and **204** connected to system bus **206**. Alternatively, a single processor system may be employed. Also connected to system bus **206** is memory
15 controller/cache **208**, which provides an interface to local memory **209**. I/O bus bridge **210** is connected to system bus **206** and provides an interface to I/O bus **212**. Memory controller/cache **208** and I/O bus bridge **210** may be integrated as depicted.

20 Peripheral component interconnect (PCI) bus bridge **214** connected to I/O bus **212** provides an interface to PCI local bus **216**. A number of modems may be connected to PCI local bus **216**. Typical PCI bus implementations will support four PCI expansion slots or add-in connectors.

25 Communications links to clients **108-112** in **Figure 1** may be provided through modem **218** and network adapter **220** connected to PCI local bus **216** through add-in boards.

Additional PCI bus bridges **222** and **224** provide interfaces for additional PCI local buses **226** and **228**,
30 from which additional modems or network adapters may be supported. In this manner, data processing system **200**

Docket No. AUS920010942US1

allows connections to multiple network computers. A memory-mapped graphics adapter **230** and hard disk **232** may also be connected to I/O bus **212** as depicted, either directly or indirectly.

5 Those of ordinary skill in the art will appreciate that the hardware depicted in **Figure 2** may vary. For example, other peripheral devices, such as optical disk drives and the like, also may be used in addition to or in place of the hardware depicted. The depicted example is
10 not meant to imply architectural limitations with respect to the present invention.

The data processing system depicted in **Figure 2** may be, for example, an IBM e-Server pSeries system, a product of International Business Machines Corporation in
15 Armonk, New York, running the Advanced Interactive Executive (AIX) operating system or LINUX operating system.

With reference now to **Figure 3**, a block diagram illustrating a data processing system is depicted in which
20 the present invention may be implemented. Data processing system **300** is an example of a client computer. Data processing system **300** employs a peripheral component interconnect (PCI) local bus architecture. Although the depicted example employs a PCI bus, other bus
25 architectures such as Accelerated Graphics Port (AGP) and Industry Standard Architecture (ISA) may be used. Processor **302** and main memory **304** are connected to PCI local bus **306** through PCI bridge **308**. PCI bridge **308** also may include an integrated memory controller and cache
30 memory for processor **302**. Additional connections to PCI local bus **306** may be made through direct component interconnection or through add-in boards. In the depicted

Docket No. AUS920010942US1

example, local area network (LAN) adapter **310**, SCSI host bus adapter **312**, and expansion bus interface **314** are connected to PCI local bus **306** by direct component connection. In contrast, audio adapter **316**, graphics adapter **318**, and audio/video adapter **319** are connected to PCI local bus **306** by add-in boards inserted into expansion slots. Expansion bus interface **314** provides a connection for a keyboard and mouse adapter **320**, modem **322**, and additional memory **324**. Small computer system interface (SCSI) host bus adapter **312** provides a connection for hard disk drive **326**, tape drive **328**, and CD-ROM drive **330**. Typical PCI local bus implementations will support three or four PCI expansion slots or add-in connectors.

An operating system runs on processor **302** and is used to coordinate and provide control of various components within data processing system **300** in **Figure 3**. The operating system may be a commercially available operating system, such as Windows 2000, which is available from Microsoft Corporation. An object oriented programming system such as Java may run in conjunction with the operating system and provide calls to the operating system from Java programs or applications executing on data processing system **300**. "Java" is a trademark of Sun Microsystems, Inc. Instructions for the operating system, the object-oriented operating system, and applications or programs are located on storage devices, such as hard disk drive **326**, and may be loaded into main memory **304** for execution by processor **302**.

Those of ordinary skill in the art will appreciate that the hardware in **Figure 3** may vary depending on the implementation. Other internal hardware or peripheral

Docket No. AUS920010942US1

devices, such as flash ROM (or equivalent nonvolatile memory) or optical disk drives and the like, may be used in addition to or in place of the hardware depicted in **Figure 3**. Also, the processes of the present invention
5 may be applied to a multiprocessor data processing system.

As another example, data processing system **300** may be a stand-alone system configured to be bootable without relying on some type of network communication interface,
10 whether or not data processing system **300** comprises some type of network communication interface. As a further example, data processing system **300** may be a personal digital assistant (PDA) device, which is configured with ROM and/or flash ROM in order to provide non-volatile
15 memory for storing operating system files and/or user-generated data.

The depicted example in **Figure 3** and above-described examples are not meant to imply architectural limitations. For example, data processing system **300**
20 also may be a notebook computer or hand held computer in addition to taking the form of a PDA. Data processing system **300** also may be a kiosk or a Web appliance.

Figure 4 shows how a Java applet is handled using a Java Virtual Machine (JVM) that implements the Java 2 SDK
25 security architecture. As shown in **Figure 4**, a web page **410** may include a hyperlink, or the like, to a Java applet **420**. When a user of the web browser **430** selects the hyperlink, or otherwise initiates the download of the applet **420**, the applet code is fetched from the server
30 **415** associated with the applet **420**. The fetched code is verified by a byte-code verifier **440** and the applet is instantiated as a class or set of classes **450** in a

Docket No. AUS920010942US1

namespace **460** by a class loader **445**.

At the time the classes **450** of the applet are instantiated, the JVM **470** also builds a protection domain for the applet **420**. The protection domain is a data
5 structure in which a set of permission collections are present. A permission collection is a grouping of permissions defined by a developer. Every time a developer defines a new permission, the developer must also define a permission collection to which the new
10 permission belongs or assume the default implementation. These permission collections may be assigned to classes **450** of the applet **420**. Thus, when the classes are instantiated, the JVM looks at the permission collections assigned to the classes and generates a protection domain
15 based on the assigned permission collections.

The bytecode is then executed by the JVM **470**. The bytecode is executed as threads of execution. A thread is a stream of execution. Threads allow multiple streams of execution to occur virtually simultaneously in a data
20 processing system thereby allowing multitasking. An executing thread has a thread stack. The thread stack is a mechanism for tracking which method calls which other method in order to be able to return to the appropriate program location when an invoked method has finished its
25 work.

During execution, the bytecode may make calls to potentially dangerous, or untrusted, functionality. When such a call is made by a thread of execution, the SecurityManager **480** calls a
30 SecurityManager.checkPermission() method which in turn calls an AccessController.checkPermission() method of the Access Controller **485**.

Docket No. AUS920010942US1

The SecurityManager **480** is the part of a JVM that enforces the security policy of the computing system on which the JVM is resident. When an untrusted operation is to be performed by an application, the SecurityManager

5 **480** is responsible for checking whether the application has the appropriate permission for performing the operation.

A permission represents access to a resource. In order for a resource access to be allowed, the

10 corresponding permission must be explicitly granted to the code attempting the access. A permission typically has a name and, in some cases, a comma-separated list of one or more actions. For example, the following code creates a FilePermission object representing read access

15 to the file named abc in the /tmp directory:

```
perm = new java.io.FilePermission("/tmp/abc","read");
```

In this permission, the target name is "/tmp/abc" and the

20 action string is "read".

It is important to note that the above statement creates a permission object that represents, but does not grant access to, a system resource. Permission objects are constructed and assigned, or granted, to code based

25 on the policy in effect. When a permission object is assigned to some code, that code is granted the permission to access the system resource specified in the current security manager when making access decisions. In this case, the (target) permission object is created

30 based on the requested access, and checked against the permission objects granted to and held by the code making the request.

Docket No. AUS920010942US1

The security policy for an application environment is represented by a Policy object. In the default implementation, PolicyFile, the policy can be specified within one or more policy configuration files. The
 5 policy file(s) specify what permissions are allowed for code from specified code sources. A sample policy file entry granting code from the /home/sysadmin directory read access to the file /tmp/abc is:

```
10      grant codeBase "file:/home/sysadmin/" {
          Permission java.io.FilePermission "/tmp/abc",
          "read";
      };
```

In the Java Development Kit 1.1, it was the
 15 responsibility of the SecurityManager to directly call a check() method on untrusted resource access requests in order to determine if the resource access request should be granted. In Java 2 SDK, it is the AccessController
 485 that calls the AccessController.checkPermission() method on permission objects.
 20

When the AccessController.checkPermission() method is called by the AccessController 485 on a permission object, the AccessController 485 retrieves the AccessControlContext for a thread of execution that
 25 resulted in the call of the AccessController.checkPermission() method. The AccessControlContext is a combination of an array of protection domains for the classes in the thread stack, and a CodeSource. The CodeSource is a combination of an
 30 origination location of a resource access request and a set of zero or more digital signatures.

Having retrieved the AccessControlContext, the

Docket No. AUS920010942US1

Access Controller **485** calls an
AccessControlContext.checkPermission() method on the
AccessControlContext. The
AccessControlContext.checkPermission() method calls an
5 AccessControlContext.implies() method which in turn calls
implies() on each protection domain identified in the
AccessControlContext to determine if the particular
permission being checked is present in each of the
ProtectionDomain objects. This causes an implies()
10 method to be called on the java.security.Permissions
object in each of the ProtectionDomains. This, in turn,
causes an implies() method to be called on each
permission in each PermissionCollection specific to that
type. In this way, each permission in each relevant
15 PermissionCollection of each ProtectionDomain identified
in the AccessControlContext is checked to see if it
corresponds to the permission being checked.

If the results of this check indicate that any one
of the protection domains does not include the requisite
20 permission, i.e. the permission being checked, then the
requested resource access is denied. Thus, all
protection domains identified by the AccessControlContext
must include the permission being checked in order for
the access request to be granted. This enforces the
25 requirement that each ProtectionDomain include at least
one permission collection that includes the permission
being checked.

Thus, in known systems, a developer of an
application must possess a sufficient amount of knowledge
30 about the security policy and permissions of a particular
computing system in order to make sure that the new
application may work with the existing system and system
resources. The application developer must know each new

Docket No. AUS920010942US1

permission and identify a PermissionCollection to which the new permission belongs. The present invention provides a mechanism by which such detailed knowledge of the security policy and permissions is not required and applications may be installed with relative ease.

With the present invention, permissions are defined in a hierarchical manner such that there are superclass permissions and subclass permission of the superclass permission. The result is a tree-like structure of permissions that may be assigned to classes of installed applications.

Figure 5 is an exemplary diagram of a permissions hierarchy according to the present invention. As shown in **Figure 5**, a superclass permission is defined as IBMPermission. This superclass permission has a plurality of subclass permissions including CorePermission and WSPermission. In turn, these subclass permissions may have further subclass permissions including Permission1, Permission2 and Permission3. The subclass permission WSPermission is a superclass permission to Permission1, Permission2 and Permission3.

With the present invention, when a class is assigned a superclass permission, all subclass permissions of the superclass permission are also allocated to the installed class. For example, if the superclass permission WSPermission is assigned, or granted, to an installed class, the subclass permissions Permission1, Permission2 and Permission3 are also allocated to the installed class. In this way, the developer need only know about the superclass permission and grant the superclass permission to the installed class in order to obtain the benefit of all of the subclass permissions.

Docket No. AUS920010942US1

This is especially beneficial for applications that are part of the same overall product. If a developer adds an application to an existing product, for example, the developer need only assign a superclass permission of an existing permission hierarchy of the product to the new application. In this way, the application developer need not know the detailed layout of the existing permissions and yet obtain the benefit of existing permissions by inheritancy through the permissions hierarchy.

The present invention provides the ability to allocate all subclass permissions of a superclass permission to an installed class of an application by placing these permissions into the PermissionCollection of the superclass permission. This may be done in a number of different ways, two of which will be described in detail hereafter. These two mechanisms for placing subclass permissions into the PermissionCollection of the superclass permission are preferred embodiments, however, the invention is not limited to such. Rather, any mechanism for allocating subclass permissions of a superclass permission that is granted to an installed class may be used without departing from the spirit and scope of the present invention.

Figure 6 illustrates a first mechanism for performing the operations of the present invention. As shown in **Figure 6**, when an untrusted resource access request is received from the bytecode **610**, the JVM **620** invokes the SecurityManager **630** which determines the required permission based on the CodeSource and the resource. The SecurityManager **630** then calls a SecurityManager.checkPermission() method on the

Docket No. AUS920010942US1

identified permission. This call results in a call of the AccessController.checkPermission() method which causes the AccessController **640** to retrieve the AccessControlContext for the execution thread and call
5 the AccessControlContext.checkPermission() method on the AccessControlContext.

When the Access Controller calls the AccessControlContext.checkPermission() method, an implies method is called on each permission **650** in each relevant
10 PermissionCollection of each ProtectionDomain in the AccessControlContext **660**.

With this particular embodiment of the present invention, however, each permission is provided with a method called newPermissionCollection. The results of
15 the calling of the implies method on the permission are input to the newPermissionCollection method. The newPermissionCollection method determines, based on these results, the superclass permission of the permission being checked and whether the superclass permission is
20 present in each of the protection domains in the AccessControlContext stack.

If so, the permission being checked is added to the new PermissionCollection. The new PermissionCollection is then added to the AccessControlContext **660**, if the
25 permission check passes then control is given back to the SecurityManager, otherwise an exception is thrown. If, however, the superclass permission has not been granted, the resource access request is denied and the JVM **620** informs the bytecode **610** of the denial of access to the
30 system resource and a SecurityException is thrown.

In an alternative embodiment, rather than checking the superclass permission to see if it has been granted

Docket No. AUS920010942US1

and then adding only the permission being checked to the newPermissionCollection, the present invention may operate to add all subclass permissions of the checked permission to the new PermissionCollection. That is, the
5 newPermissionCollection method may determine whether the superclass permission is present in each of the protection domains and if so, add the checked permission and any subclass permissions of the checked permission, as determined from the security policy, to the new
10 PermissionCollection.

Figures 7A and 7B are exemplary diagrams of a superclass permission in accordance with the present invention. **Figure 7C** is an exemplary diagram of a subclass permission according to the present invention.
15 As shown in **Figures 7A-7C**, these permissions both include calls of the newPermissionCollection method which is used by the present invention to generate and return a new PermissionCollection that includes the permission being checked for and any subclass permissions of the
20 permission being checked for. In addition, the subclass permission shown in **Figure 7C** includes identification that the subclass permission WSPermission is a subclass of the superclass IBMPermission shown in **Figures 7A and 7B**.

Figure 8 is an exemplary illustration of code for performing the operations of the present invention using the superclass permission IBMPermission and subclass permission WSPermission shown in **Figures 7A and 7B**. The sequence of events for checking a particular permission,
25 as shown in **Figure 8**, is as follows. First, the security manager calls a SecurityManager.checkPermission method on the permission. The call of the
30

Docket No. AUS920010942US1

SecurityManager.checkPermission method causes an
AccessController.checkPermission method to be called on
the permission.

The current AccessControlContext is then retrieved
5 and an AccessControlContext.checkPermission method is
called on the permission. The
AccessControlContext.checkPermission method calls an
AccessControlContext.implies method on the
AccessControlContext. The AccessControlContext.implies
10 method calls a ProtectionDomain.implies method on the
ProtectionDomain object.

The AccessControlContext.implies method calls a
PermissionCollection.implies method on the
PermissionCollection objects that are encapsulated in the
15 ProtectionDomain. The PermissionCollection objects are
obtained from the current security policy. It is then
determined whether the PermissionCollection associated
with the permission is present in the ProtectionDomain.
If so, the PermissionCollection is returned. Otherwise,
20 a new PermissionCollection is constructed by a call to
the newPermissionCollection method. As shown in
Figure 8, when some code is granted IBMPermission, the
PolicyFile instantiates a ProtectionDomain (referred to
as "pd") that contains an IBMPermission object as part of
25 an IBMPermissionCollection (which is in the Permissions
object referred to as "perms"). When a call to Security
Manager.checkPermission method is made, a permissions
object implies method is called. At this point, the
permission objects looks into an internal hashtable to
30 see what PermissionCollection must be associated with the
permission so that it can also see if that
PermissionCollection.implies the permission or not.
However, the permissions object does not know which

Docket No. AUS920010942US1

PermissionCollection must be associated with the permission (it does know which PermissionCollection must be associated with an IBMPermission since an IBMPermission was specified in the security policy file, but it does not know about the permission yet). Therefore, it calls the newPermissionCollection method, which returns an IBMPermissionCollection object. This IBMPermissionCollection object is, empty and will not imply the WSPermission. Thus, a security exception will be thrown.

Figure 9 is a flowchart outlining an exemplary operation of the present invention. As shown in **Figure 9**, the operation starts with an untrusted resource access request being received (step **910**). The required permission for the access request is determined based on the CodeSource and the resource (step **920**). The SecurityManager.checkPermission method is called on the required permission (step **930**). This causes an AccessController.checkPermission call which in turn calls an AccessControlContext.checkPermission call on the AccessControlContext for the thread of execution originating the resource access request (step **940**).

The AccessControlContext.checkPermission call results in an implies method being called on each permission of each PermissionCollection in the ProtectionDomains of the AccessControlContext (step **950**). The newPermissionCollection method of each permission is then called (step **960**).

The newPermissionCollection method makes a determination as to whether a superclass permission of the required permission is present in each of the ProtectionDomains of the AccessControlContext (step **970**).

Docket No. AUS920010942US1

If not, the resource access request is denied, an exception is thrown (step 975) and no further processing occurs ends.

5 If the superclass permission is present in each of the ProtectionDomains, the required permission is added to the new PermissionCollection (step 980). In an alternative method, any subclass permissions of the required permission may also be added to the PermissionCollection (step 985, shown in dotted lines to
10 indicate an alternative embodiment). The PermissionCollection is then added to the AccessControlContext (step 990) and the resource access request is granted (step 995).

15 In an alternative embodiment, the functionality described above may be hardcoded into the security policy file class such that modification of permissions to include the newPermissionCollection method is not necessary. Rather, the functions of the newPermissionCollection method may be hardcoded into the
20 security policy file class such that the code operates on all permissions processed using this security policy class.

In this alternative embodiment, a subclass Policy of the superclass PolicyFile is created such that its
25 getPermissions() method takes the Permissions object returned by the getPermissions() method of the superclass PolicyFile and, before returning it, determines if the Permissions object implies the superclass permission of the permission being checked , i.e. the superclass
30 permission is present in each of the protection domains.

If the Permissions object implies the superclass permission, then the permission being checked is added to

Docket No. AUS920010942US1

the permission collection of the Permissions object and the resource access request is granted. If the Permissions object does not imply the superclass permission, the permission being checked is not added to the PermissionCollection of the Permissions object and the resource access request is denied with an exception thrown.

Figure 10 is a flowchart outlining an exemplary operation of the present invention according to this alternative embodiment. As shown in **Figure 10**, the operation starts with an untrusted resource access request being received (step **1010**). The required permission for the access request is determined based on the CodeSource and the resource (step **1020**). A determination is made as to whether the required permission implies the superclass permission of the required permission (step **1030**).

If the permissions object implies the superclass permission, then the permission being checked is added to the permission collection of the Permissions object (step **1040**) and the resource access request is granted (step **1050**). If the Permissions object does not imply the superclass permission, the permission being checked is not added (step **1060**) to the PermissionCollection of the Permissions object and the resource access request is denied with an exception thrown (step **1070**).

Figure 11 provides code for performing the operations according to this alternative embodiment. It should be noted that, for simplicity, the name of the IBMPermission subclass, WSPermission, is hardcoded in this PolicyFile subclass. In a more sophisticated implementation, the names of the subclasses may be

Docket No. AUS920010942US1

hardcoded in a signed configuration file, and the PolicyFile subclass may check the signature before accepting to read that file and retrieve the names of the subclasses from it.

5 Thus, the present invention provides mechanisms by which a developer may grant a superclass permission to an installed class and obtain the benefit of subclasses without having to know the details of the security policy. This allows development and integration of
10 applications into existing systems to be done seamlessly and with greater ease.

It is important to note that while the present invention has been described in the context of a fully functioning data processing system, those of ordinary
15 skill in the art will appreciate that the processes of the present invention are capable of being distributed in the form of a computer readable medium of instructions and a variety of forms and that the present invention applies equally regardless of the particular type of
20 signal bearing media actually used to carry out the distribution. Examples of computer readable media include recordable-type media, such as a floppy disk, a hard disk drive, a RAM, CD-ROMs, DVD-ROMs, and transmission-type media, such as digital and analog
25 communications links, wired or wireless communications links using transmission forms, such as, for example, radio frequency and light wave transmissions. The computer readable media may take the form of coded formats that are decoded for actual use in a particular
30 data processing system.

The description of the present invention has been presented for purposes of illustration and description, and is not intended to be exhaustive or limited to the

Docket No. AUS920010942US1

invention in the form disclosed. Many modifications and variations will be apparent to those of ordinary skill in the art. The embodiment was chosen and described in order to best explain the principles of the invention,
5 the practical application, and to enable others of ordinary skill in the art to understand the invention for various embodiments with various modifications as are suited to the particular use contemplated.